Collapsing the Vertical Stack

Deterministic State Management from Web APIs to Silicon

Scott McLeod
McLeod Interactive Group LLC · Jacksonville, Florida

Version 1.0 · June 2026

ABSTRACT

Every system that drives a group toward a decision rebuilds the same machinery: surface options, collect preferences, aggregate them, route the result to whoever holds authority, and close the loop. The machinery is rebuilt badly because it conflates two things that should stay separate: the *arbitration* of what should happen, and the *authority* to make it happen. This paper presents **Cooren**, a coordination primitive of six domain-agnostic operations governed by a single gated state-transition function in which authority is binary, default-off, and held outside the system being arbitrated. The defining property is that consensus is necessary but never sufficient: with the gate closed, the next state equals the current state, and no degree of agreement or model confidence executes a change. The primitive was extracted from a running consumer application, not theorized; the same six operations are demonstrated running across three substrates: a REST API, a production mobile app, and microcontroller firmware. The paper is deliberate about the line between what runs and what is specified: the strongest claim, that the authority boundary is enforced by hardware and cannot be bypassed by the code it governs, is specified here and reported as forthcoming. We address the algebraic equivalence to a gated recurrent unit directly: the form was derived from the artifacts and converges on a known result rather than descending from it, and the contribution is the interpretation and its generalization. We frame a bounded slice of the alignment problem, unauthorized action and authority capture, as a security property with a defined attack surface.

01 The plumbing nobody factored out

A family deciding where to eat, four AI agents resolving a logistics exception, and an industrial controller arbitrating sensor inputs against a safety limit are, structurally, the same problem. Each opens a context, registers the parties with something to say, collects what they signal, aggregates it, routes the aggregate to whoever is allowed to decide, and informs everyone of the outcome. This shape recurs everywhere coordination happens, and almost everywhere it is re-implemented from scratch, fused into the surrounding application and entangled with the thing that should stay separable from it: the authority to act on the result.

The entanglement is the defect. When the logic that *computes* a recommended action lives in the same trust domain as the logic that *commits* it, "the system did something no one authorized" is not a violation the architecture can name. It is just an outcome. Polling tools, consensus libraries, and agent frameworks all compute aggregates well. None of them treats the authority to execute as a first-class, externally-held object distinct from the aggregate itself.

Foundational infrastructure is rarely launched. It is documented, implemented by someone else, and named later. The aim here is narrow. It does not market a product. It specifies a primitive precise enough to implement, shows the same primitive already running at three layers of the stack, and states without ornament where the work is finished and where it is not.

02 The primitive: six operations

Cooren is six operations. They carry no knowledge of their domain. They move participants, signals, tallies, and decisions, and the caller supplies what those mean. A thermostat zone, a freight dispatcher, and an AI agent are all participants that submit signals. The engine does not know which is which.

#	OPERATION	ROLE
1	`session_create`	Open an isolated coordination context.
2	`participant_register`	Register an input or endpoint; issue it an identity.
3	`signal_submit`	A participant submits a value and a priority.
4	`tally_get`	Aggregate the submitted signals into a single view.
5	`decision_record`	Bind the authoritative output state.
6	`announcement_send`	Dispatch the decision and dissolve the loop.

The primitive was discovered, not invented. It was extracted from The Dinner Decider, a family meal-voting application running in production on the Apple App Store and Google Play. The coordination logic that made that app work proved to be entirely domain-agnostic; stripped of the dinner context, what remained was infrastructure. That provenance matters: the six operations are a generalization of code that already shipped, not a clean-room abstraction in search of a use.

03 The gate equation

The six operations are sequenced by one transition function. It states the thesis in one line.

$$S(t+1) = G(t) \cdot A(\, S(t), I(t)\,) + (\, 1 - G(t)\,) \cdot S(t)$$

when $G(t) = 0 \rightarrow S(t+1) = S(t)$

S = system state · I = aggregated signals · A = arbitration · G = the authority gate

A is the arbitration function: it reads the current state and the aggregated signals and proposes the next state. G(t) is the authority gate. It is **binary**: it opens or it does not, and it is set **outside** A, by an authority the arbitrating system does not control. When the gate is open, the proposal becomes the new state. When it is closed, the state holds unchanged, regardless of how the signals tallied or how confident the arbitration was.

The reduction line is the point. The default behavior of the system is *persistence*, not action. A unanimous tally with maximum confidence and every threshold satisfied still produces no state change while the gate is closed. This is sometimes called fail-toward-stillness: the system's resting state is to do nothing, and a positive act of authority is required to leave it.

Consensus is necessary. Consensus is never sufficient*.*

Constraints and Parameters: the content of the gate

The gate does not open arbitrarily. Its output G(t) is the result of evaluating two kinds of pre-committed authority against the proposed transition. **Constraints** are hard, inviolable boundaries; a single violation forces G(t)=0 and escalates to a human or a safe state. **Parameters** are tunable thresholds inside the legal space, for example a quorum or a confidence floor, that set how readily the gate opens. Constraints and Parameters are authority encoded in advance: they let the gate enforce the holder's intent on every cycle without the holder present, and they hand control back to a human only on escalation, when they cannot resolve a case on their own.

04 Provenance and the gated-recurrent-unit equivalence

The equation was not the starting point. The primitive was built and observed at four levels before the equation existed: a consumer application, a REST API, a multi-agent orchestration demo, and microcontroller firmware. The equation is the compression of what those implementations already had in common, derived from the artifacts rather than imported from an external model.

That derived form is algebraically equivalent to the update step of a gated recurrent unit, which mixes a previous hidden state with a candidate through an update gate z: $h(t) = z \cdot \tilde{h} + (1 - z) \cdot h(t-1)$. The equivalence is stated here plainly, because a careful reader will reach for it, and because arriving at a known-good form independently is a stronger position than inventing it or taking it from elsewhere. It is convergent evidence that the form is fundamental to gated coordination, not an artifact of one domain.

The contribution is the **interpretation**, and it inverts three properties of the GRU gate. In a recurrent unit the gate `z` is *learned*, *internal*, and *continuous*: a differentiable parameter of the network it governs, trained to whatever value minimizes loss. Cooren's gate is *authored*, *external*, and *binary*. It is not a parameter the system tunes about itself. It is a boundary set by an authority the system cannot reach, default-closed, opened only when pre-committed Constraints and Parameters are satisfied under that authority. A GRU learns how much of its proposal to let through. Cooren asks who is permitted to let it through at all.

The second half of the contribution is generalization. The same interpretation holds across substrates that share no implementation. A database transaction, an agent pipeline, and a memory-protection fault are three enforcement mechanisms for one gate. The form coincides with prior art, which is acknowledged. The contribution is the default-off, external-authority reading of it, carried unchanged from cloud to silicon.

05 Substrate invariance: what is running

The demonstrated claim is narrow and verifiable: the same six operations, with the same semantics, run on three substrates that share no code. This is *substrate* invariance: one primitive across many enforcement layers. It is distinct from the broader *domain* invariance the primitive also exhibits across application areas, which is a separate axis and a softer claim.

SUBSTRATE	IMPLEMENTATION	WHAT IT PROVES	STATUS
REST API	PHP 8.4 / MariaDB, six operations exposed as HTTP endpoints at `api.cooren.dev`. Gate enforced by the transaction boundary.	The primitive as a stateless network service.	RUNNING
Consumer app	The Dinner Decider, live on App Store and Google Play; the production origin of the primitive.	The primitive end-to-end, with real users and an authority holder.	RUNNING
Firmware	C++ on AVR (Mega 2560) and RP2350 (Pico 2). The coordination core carries no pin references and compiles unchanged from the API's logic.	The loop on bare metal: six ops, consensus dwell, gated decision.	RUNNING
Silicon boundary	RP2350 MPU: the gate runs privileged, participants run unprivileged, the decision buffer is privileged-only memory.	That the gate is *physically* unbypassable. See §6.	FORTHCOMING

SPEC vs RUNNING. *The firmware loop is verified in simulation and on the loop level. Simulation proves the loop: six operations executing in order, signals dwelling to consensus, the decision gated. It proves nothing beyond that, and does not model the memory-protection boundary of §6. The claim made for the firmware row is exactly "the loop runs on silicon," not "the boundary is enforced by hardware." The latter is the next section, and it is not yet true.*

06 The silicon boundary FORTHCOMING

In software, the gate is enforced by code that sits in the same trust domain as the participants it governs. A determined participant could, in principle, rewrite the Constraints that bind it. The strongest form of the thesis requires removing that possibility physically rather than by convention.

On the RP2350, the design places the gate in **privileged** code behind the Cortex-M33 memory protection unit. The gate is the tally evaluation, the Constraint and Parameter checking, `decision_record`, and `announcement_send`. Participant tasks run **unprivileged**. The pending-decision buffer lives in a region the MPU marks privileged-access-only. A participant may call `signal_submit`, which writes only to the signal buffer; any attempt by an unprivileged task to write the decision path directly raises a hardware MemManage fault.

```
// the proof: a participant tries to reach the decision path
void MemManage_Handler(void) {
    g_gate_violations++;          // the gate held
    // input could not become decision without
    // crossing the privileged boundary
}
```

That fault is the entire point of going to silicon. When the violation counter increments because an unprivileged task was denied the decision path, that is G(t) with teeth. The chip, not a trust assumption, makes it impossible for a signal to become a decision without crossing the authority boundary.

Honest status. *This boundary is specified and not yet running. None of it is claimed as demonstrated. When the fault fires on hardware, it will be reported as a dated version-2 addendum to this deposit, under the same concept identifier. Nothing in this paper should be read as asserting the boundary holds today. What holds today is the loop. The proof that the loop cannot be circumvented is the work that remains.*

07 Alignment as a security property

A precise claim, stated with its own limits attached. Cooren does **not** resolve the intrinsic alignment problem. It says nothing about reward hacking, goal misgeneralization, or deceptive alignment, and it offers no method for making an arbitration function *want* the right thing. Any claim to the contrary would be an overclaim, and this paper does not make it.

What the primitive addresses is a different and narrower failure: **unauthorized action**. When authority is a gate external to the arbitrating system and default-off, the event "an agent took an action no one authorized" stops being an emergent behavior to be trained away and becomes a *security violation with a defined surface*. The arbitration may be wrong, biased, or adversarial. It still cannot move the system's state, because moving state requires the gate, and the gate is not its to open. The attack surface contracts to the authentication of the gate itself, a boundary that can be audited, hardened, and, as §6 argues, enforced in hardware.

This is the complete-mediation principle applied to autonomy: every state transition passes the gate, with no path around it. It converts one bounded slice of alignment, authority capture and unauthorized

execution, from an open research problem into an access-control problem. Access control is a category engineering already knows how to reason about. The rest of alignment remains exactly as hard as it was.

08 What is open and what is held

The primitive is published to be implemented. The boundary around what is published is itself an architectural decision, and it is shaped like public-key infrastructure: **open the verifier, hold the issuer.**

Open. The six operations and their semantics; the gate equation and its default-off reading; the Cooren Signal Protocol wire format, which carries coordination state in every frame: who is participating, what they signal, and under whose authority; the chain-of-custody format by which any frame's authority lineage can be walked back and verified, failing at exactly the link that is forged or broken; and a stubbed reference implementation. Anyone can stand up an engine, run the loop, and verify a chain of authority end to end.

Held. The origination of the authority anchor and its key-derivation, and the intent-bound enrollment by which a new authority holder is admitted. These are withheld not as a proprietary reflex but as the security boundary that makes the open part safe to publish. A verifier anyone can run, combined with enrollment no one can forge, is the structural defense against the authority layer being captured. The reader can confirm that a chain is legitimate without acquiring any ability to mint a false one. Openness is the verification surface. Sovereignty is the issuance surface. The line between them is deliberate.

09 Reference surfaces

The primitive is meant to be run, not read about. Four live surfaces and the reference implementation let a reader confirm every running claim above directly.

SURFACE	WHAT IT IS	WHERE
Coordination API	Six operations as a live REST service.	`cooren.dev`
Consumer reference	The production app the primitive was extracted from.	`thedinnerdecider.net`
Multi-agent demo	Domain-bounded agents coordinating through live sessions with human-in-the-loop escalation.	`cooren.dev/project/jeda`
Reference implementation	Spec, gate equation, wire format, stubbed engine.	`github.com/McLeod-Interactive-Group-LLC/cooren-api`

The reference repository is the canonical implementation home. This deposit is the canonical claim of record. They are separate by design: the code is run, the deposit is cited.

10 Conclusion

This is not a request for permission. It is the publication of a pattern that already runs at three layers of the stack, reduced to a contract small enough to implement and precise enough to test. The primitive's value is that it gives a name and a boundary to the one thing every coordinating system needs and almost none make explicit: a gate that separates deciding what should happen from holding the authority to make it happen, defaulting closed, set from outside.

The honest edges are marked throughout. The equation was derived from the artifacts and converges on prior art rather than descending from it. The interpretation is the claim. The loop runs on silicon, and the proof that it cannot be bypassed is specified and forthcoming. A bounded slice of alignment is recast as access control, and the rest is left untouched. The single line in the whole program that can actually fail is not in this paper. It is one external operator running one real decision, end to end, on their own system, and choosing to run it again.

Hold Fast · Find A Way · Shine Brightly

TITLE	Collapsing the Vertical Stack: Deterministic State Management from Web APIs to Silicon
AUTHOR	Scott McLeod · McLeod Interactive Group LLC
VERSION	1.0 · June 2026 · v2 addendum to follow on silicon-boundary completion
DEPOSIT	Zenodo, canonical deposit · DOI registers on publication
LICENSE	Paper: CC BY 4.0 · Reference implementation: permissive (see repository)
REPOSITORY	github.com/McLeod-Interactive-Group-LLC/cooren-api
CONTACT	mcleodinteractivegroup.com · cooren.dev